SUPERKIT 1541

Has it all!

BY MARTY FRANZ & JOE PETER

USER GUIDE

written by Jim Ray

INTRODUCTION

The programs contained on the Super Kit disk arm the user with some of the most powerful and advanced DOS utilities available to date. They were designed and incorporated to benefit the least experienced as well as the high-level programmer. The Super Kit programs are written entirely in machine language.

These programs have been extensively tested and found to work properly. Certainly, it is possible there could be an occasional problem, or merely some aspect that could be improved, which occurs in your particular application. Please feel free to contact us with your problems or suggestions. In this way we can work to make a better product to benefit every user.

Although the utilities in Super Kit were developed to facilitate duplication of DOS-protected disks, there are several utilities which will benefit a programmer's day to day tasks. You will find Super Kit crammed to the brim, both front and back, with the tools to get that difficult program or disk reproduced for your archives.

NOTICE OF COPYRIGHT

NOTICE

TABLE OF CONTENTS

Introduction..... 2
Getting started..... 5
Program selection shortcuts..... 5
Initial Menu..... 6
A word about the programs..... 7
Using the file copier..... 11
The Super Kit sector editor..... 15
  Main commands..... 17
  Monitor commands..... 17
The Super Kit GCR editor..... 20
  Main menu commands..... 21
  Data editor commands..... 22
Super Dos (tm)..... 25
Autoboot creator..... 27
Super Kit Sector Surgeon..... 28
Super Scan..... 29
Appendix..... 31

The History of Commodore 64 Program Protection
by Rob Vaughn

Chapter 1 - Random files, disk data, disk errors, data under errors..... 34
Chapter 2 - Extra tracks, half-tracks, embedded tracks and DOS manipulation..... 39
Chapter 3 - Sync manipulation, extended data and header blocks..... 45
Chapter 4 - Group Code Recording (GCR)..... 53

GETTING STARTED

The Super Kit diskette contains program material on both sides of the disk. The programs are designed to be used with the Commodore 1541 and the newer 1571 disk drives. Swiftest results in program duplication will be obtained using two of these drives (in any combination).

LOADING. Either side of your Super Kit diskette may booted with LOAD ":*",8,1. Be sure that you have removed FAST LOAD, MACH-5, ANY CARTRIDGE OR DEVICE from the game port before attempting to load the programs. A fast-booter will be engaged to promptly load the program.

We suggest you first review this user guide, then load the Super Nibbler and make a backup copy of both sides of Super Kit. Now store your original in a safe place and use the copied versions for future tasks. The archival versions will now be "customized" to your drive for optimum results.

SHORTCUT TO PROGRAM LOADING. After you become more acquainted with the programs in your Super Kit, you may wish to shorten your loading time and go directly to a certain program (such as an editor). Screen displays can be shortened by pressing the spacebar. Any file on the menus can be loaded individually in the following manner, LOAD "*x",8,1 (where x is the number of the program you want to load). Those programs by number are:

SUPER KIT - front side of diskette.

1 - Dual Normal Disk Copy
2 - Single Normal Disk Copy
3 - Dual Fast Nibble Copy
4 - Single Fast Nibble Copy
5 - Track and Sector Editor
6 - G C R Editor
7 - File Copier

SUPER KIT - reverse side of diskette.

1 - Super Nibbler Disk Copy
2 - Super Disk Surgeon
3 - Super Scan

There are some additional utilities on the front side of the disk which do not appear on the menus and may be viewed in the directory (Load "$",8 and LIST). These programs consist of an autoboot generator and three versions of Super Dos. These will be covered in later sections of the manual.

INITIAL MENU

On side one of Super Kit you are presented with the choice of five copy programs, a sector editor and a professional quality GCR editor (yessiree disk buffs)! Side two will give you the choice of a super nibble copier, a parameter copier and a revolutionary disk error scanner. Selection is made by moving the pointer to the appropriate choice with the function keys. F3 moves the arrow down, F1

moves it up and pressing F5 will load the selection into the computer.

A WORD ABOUT THE PROGRAMS

DUAL NORMAL. Duplicates non-protected disks and will reproduce some Dos errors (21, 22, and 29) if encountered. This version is for two disk drives and is very fast (32 seconds is average), speed of copy varying with the amount of data on the disk.

SINGLE NORMAL. This is the single drive version of the Dual Normal copier. It is also very fast requiring 3 swaps of the source disk. It does not reproduce any Dos errors but does not "crash" if any are encountered.

DUAL NIBBLER. A full disk copier for light to medium duplication of DOS protected disks. A state-of-the-art copier comparable to most nibble copiers available to date. Requires two disk drives and is very fast, copy speed varying according to amount of data contained on the disk being duplicated.

SINGLE NIBBLER. The single drive version of the Dual Nibbler that does the same job, just takes a bit longer depending on how fast you can swap the disks.

FILE COPIER. A two drive program that moves those individual files to that disk you really wanted them on to begin with, reliably, and SUPERFAST! Transfers single or multiple files of your choice

with your initial selection, even the DELETED ones (should you want to resurrect them). Works great with a single drive too!

SUPER NIBBLER. This one is BIG Brother to the other Super Kit nibblers. It is THE state-of-the-art full disk copier for those "impossible" duplications. With this copier the task of reproducing an original program is completely automatic. All that is required of you is to switch source and destination diskettes. You will find this is accomplished with a speed that belies the complexity of this type of duplicating program. Included is a provision to alter the track parameters to allow extending to track 40 or selective track copying.

DISK SURGEON. This utility program is a complete disk parameter copier. It contains many parameters to enable the user to quickly make an unprotected backup of the original. Refer to the sheet included in Super Kit for the names of the programs for which this utility may be used (or load the program and scroll through the list). You should find some 250 titles from which to choose. Updates, with additional parameters, will be forthcoming periodically.

SECTOR EDITOR. A track and sector utility whose powerful editor features allow you to directly access the data on your disk. Data may be read, edited, and written to your disk in decimal, hex or text modes. Also includes a machine language monitor to use with sector data. Other features include access to a disk command screen to perform

disk chores and instant access to a command help screen.

GCR EDITOR. The Group Code Recording editor is a powerful, professional quality utility that will allow you to examine your disk data at a very fundamental level. It will allow the user to directly view the sync, header block, checksums, embedded id's, bit density and other data that cannot be displayed with a track and sector editor. Moreover, this utility will allow you to view, EDIT and WRITE any of those parameters to your disk.

SUPER SCAN. This utility will allow you to perform a check on your disk to locate non-standard data. The locations containing this data, as well as the type of data, will then be reported when the scan is complete. The scan of the complete disk takes about 30 seconds. Super Scan will also report the bit density used on each track and sector. Non-standard densities will stand out like a "sore thumb". This feature is even quicker than the error scan.

SUPER DOS. Super Dos format converts your program files to a more efficient arrangement for the disk operating system to deal with. The file copy program allows the option of choosing the Super Dos mode when copying your files. Programs on disk in the Super Dos format, when used with one of the Super Dos fast-loader on this disk, will load up to FOUR times faster than the present fastload cartridges and programs. (How does 200 blocks in approximately 12 seconds grab ya?) Any files in Super Dos format will not affect normal program

loads (without engaging Super Dos) and will still load at normal speed. However, they transform into SUPERFAST loading programs with the activation of Super Dos prior to loading.

AUTOBOOT CREATOR. This program was designed to allow you to generate an autoboot file for your Basic or machine language program using one of the fastloading Super Dos options.

All of the copiers, except the file copier, allow the user to modify some parameters. The default parameters are set for optimum and duplicating a diskette is merely a matter of following the prompts. Advanced users may wish to try different settings for particularly difficult to copy programs.

After the copier is loaded, pressing the F1 key will display a screen with the following information (numbers shown are the default values for the normal copier):

F1	Starting track:	1	-variable from 1 to 40
F2	Ending track:	35	-variable from 1 to 40
F3	Track increment:	1.0	-variable from 0.5 - 2.5
F4	Sync bytes:	5	-variable from 3 to 20
F5	Header gap bytes:	8	-variable from 1 to 20
F6	Sector gap length:	9	-variable from 2 to 20
F7	Header block lgth:	8	-variable from 2 to 10
F8	Verify copy:	yes	-toggles on or off

Note that the Verify option on these copiers is a FULL verify as opposed to a TRANSFER verify used by most other fast copy programs. On the dual and single nibblers, the user can press F8 to toggle the music off. Oh yes, this one entertains while you wait.

USING THE FILE COPIER

After selecting the file copier from the main menu the program will be loaded and the user presented with a menu containing the following options:

1. Copy files
2. Scratch files
3. Directory
4. Fast format
5. Validate disk
6. View BAM
7. Change drives
8. Change mode

Also displayed are the source and destination drive numbers (defaults to a single drive 8) and the mode of operation. The user should first set up the drive configuration to be used and then select the mode in which the files are to be copied. User will also need a formatted disk to contain the copied files and may wish to take advantage of the Fast Format feature at this point.

The white selection bar can be moved with the cursor up/down key to files of the user's choice. Pressing

the return key will then select the outlined file for copying. The copier will handle any size disk file that can be contained on your disk.

COPY FILES. Insert the disk with the files you wish to copy before selecting this mode. The drive will spin and the screen will display a small menu and a list of the files contained in the disk directory. The number of files on the disk are noted on the screen. Any files selected for copying will be noted by the number and outlined in "reverse" field. The cursor up/down key will scroll the white bar through the entire directory listing.

Selecting Files. Use the cursor key to position the bar over the file wanted and press return. Any number of files may be selected in this manner. The F3 key will select and outline every file on the disk. This is useful when the user wishes to transfer all but a few of the files. Moving the cursor bar to unwanted files and pressing return will de-select those files.

Aborting the Selection Process. User can press F1 at any point and return to the main file copy menu, or press F5 which allows you to remain with the present listing.

When the files are selected, you may begin the process by pressing the 'C' key. If you are using a single drive you will be prompted to insert the source disk. If you have selected 2 drives (at the main menu) the process begins immediately.

During the copy process, if a filename is encountered that is already on your disk, you will be prompted by the program. At this point you have three options:

1) You can enter a new file name, or;
2) You can just hit RETURN and the existing file on your destination disk will be replaced with the new one, or;
3) You can press the F1 key and abort the process.

If you choose to abort, you MUST validate the destination disk before any more attempts to write to it. Also, when copying files, if a DISK FULL error message should appear you must abort and immediately validate the disk.

One final note on the copy mode. If there are any deleted files on your diskette, they will be displayed as DEL. These files may be selected and they will be copied (and restored) on the transfer disk. You will need to know what type of file is being restored before copying a deleted file. The original disk will not be affected. Use this feature WITH CARE as a deleted file will almost certainly contain corrupted blocks if the disk has been written to after the file was deleted.

SCRATCH FILES. This option deletes unwanted files from the disk in your source drive and performs in the same manner as the copy procedure. The same keys apply to the operation except that the 'S' key is used to begin the process.

DIRECTORY. Displays a listing of all the usable files on the disk in the source drive. To check for deleted files select the copy file option and you can return to the main menu by pressing F1.

FAST FORMAT. Takes approximately 10 seconds to format a disk. Enter a disk name and id at the prompts. The F1 key aborts this selection.

VALIDATE DISK. Re-organize disks to make certain all the usable space is available. Not recommended if the disk contains random files or user written data that are unallocated in BAM. A Validation will allocate the Bam correctly so this feature should be used AFTER validating.

VIEW BAM. Displays the Block Allocation Map contained on the directory track. Sectors that contain file data are displayed with a '+'. This mode also contains an editor which may be used to allocate or de-allocate any sector on the disk. Use the cursor keys to position the cursor on the desired sector(s). A block may be allocated by entering a plus '+' and unallocated by entering a minus '-' (displays as a blank sector). When you have completed your selection then press the 'W' to write it to the disk. The process may be aborted, at any time prior to writing on your disk, with the F1 key.

CHANGE DRIVES. Allows switching source and destination drives to set up one- or two-drive

8 or 9. Press return and the change will be noted by the numbers displayed by Source: and Destination: locations on the main menu. Continue to press return until the arrangement is satisfactory.

CHANGE MODE. Toggles Super DOS or Normal (default) modes for writing files.

THE SUPER KIT SECTOR EDITOR

When this selection has completed loading, you may remove your program disk and insert the disk you wish to examine. A default setting for the track and sector will be displayed. The default may be changed by entering the track you want to examine at this point, or by pressing RETURN, followed by the default sector. If you press RETURN at both prompts, the monitor will display the first sector of the disk directory track (track 18 sector 1). The default settings are the decimal value. Once into the monitor, if Hex mode is selected, choosing another track must be done in the hex values. A screen will now be displayed that is comprised of three main areas and a status display.

TEXT MODE. Top screen area, prior to reading in a sector, contains 255 '@' (hex 00) characters. This area displays the text (same characters as on your keyboard) from a chosen sector. The cursor keys are used to move the cursor to any position in this

area. The Commodore key/Shift key may be used to switch the screen into upper/lower case mode for easier reading of text that is being displayed as graphic symbols.

HEX MODE. Mid screen area, comprised of 8 columns and 7 lines of '000'. The uppermost area displays the same data as the text mode, except that the values are in hex. Since hex values are 2-digit values, disregard the first '0' in the 3-digit columns. Any hex entries will be done with 2 digits also, the keys 0-9 and A-F will function for entries in this mode.

DECIMAL EDITOR. Bottom screen area with a display like the hex mode, except that all 3-digit values will apply. In this mode, functioning keys are 0-9 (plus the Command function keys).

STATUS DISPLAY. Right side of the screen. Notes the position (POS) of the text cursor and displays the hex value. TRK is the track and SEC is the sector or block being examined. The mode by which you enter data values is indicated by the DEC (or HEX) displayed.

The cursor keys allow the positioning of the bracket-type cursors in all three of the modes. The text cursor can be moved to any byte position while the cursors in the hex and decimal modes remain in the first column. However, the data on the line scrolls beneath those cursors. All three of the cursors are in sync so that any byte within the text

cursor bracket is the same byte in the hex or decimal cursor brackets.

COMMAND HELP SCREEN. May be summoned by pressing the 'H' key. These Command Functions are entered while in the editor screen.

J - Jump to link under cursor. Allows user to position the cursor over a file or block address and it will jump to and display that block.

L - New track and sector. Use to select the track and sector you wish to examine. Entry must be value of mode (Hex or Dec) being used.

M - Activate ml monitor (with following set of commands).

A ..assembler
D ..dis-assembler
M ..display memory
P ..toggle printer on/off
X ..exit monitor
$ ..convert hex value to decimal. ascii and binary
# ..convert decimal value to hex, ascii and binary
% ..convert binary value to decimal, ascii and hex
" ..convert ascii value to decimal, hex and binary
+ ..add two hex values
- ..subtract two hex values

N - Next sector. Similar to the J command except that it is unnecessary to position the cursor over the link address to jump to the next sector.

P - Printout of viewed data.

R - Search and replace. Allows user to specify and alter any occurrence of a particular byte.

S - Speed of cursor. Sets the speed with which you can move the cursor. Ranges 01 (fastest) to 255 (slowest).

T - Type normal text. Allows editing with normal keyboard characters. Shift/T allows text to be entered in reverse characters.

V - Read previous sector. The reverse of the N command. It allows you to return to previously viewed linked sectors.

W - Write to disk. This writes any input values to your disk in the source drive.

@ - Change filler byte. Sets the value of the byte to use with the CLR command. Enter the value in hex.

CLR fills the buffer with any specified byte (default is @)

HOME moves the cursors to the start of screen data (upper left corner).

+ Move to the next adjacent sector.
- Move back to next adjacent sector.

Shift/+ and shift/- operate in the same manner except that tracks are incremented rather than sectors.

BACK ARROW (upper left on keyboard) toggles the data input mode between decimal (DEC) and hexidecimal (HEX).

F1 key selects a disk menu where you have the option to format a new disk, scratch a file, view the directory, validate your disk, or view and edit the BAM (refer to the file copy section for BAM edit commands).

A WORD OF CAUTION to inexperienced disk editor users. Altering data and using the write command will irreversibly enter the altered data on your disk. If the disk happens to be an original and the added data incorrect then you can "chalk one up" to bad judgement. Until you acquire some expertise it is recommended that you use this with copy or scratch disks.

When the Super Disk monitor is activated, the buffer data is loaded at location $2000 in the computer. The display will loop continuously from $2000 through $20FE. When viewing or disassembling the buffer data, $2000 is the address to specify but this range will be displayed regardless of any other range that you enter.

THE SUPER KIT GCR EDITOR

The Group Code Recording editor will allow you to examine your disk data at a very fundamental level. It will allow the user to directly view the sync, header block, checksum, track and sector number, embedded i.d., off bytes, header gap, data checksum and tail gap. It will also determine the header and data bit density. Moreover, the editor may be used to alter any of those parameters.

It is presumed most users of this utility will be advanced programmers well-versed in the usage of GCR. For those who wish to learn, the subject is covered elsewhere in this user guide. There are also several publications (Inside Commodore Dos by Richard Immers and Gerald Neufeld, 1541 User's Guide by Dr. Neufeld, Program Protection Manual II by CSM Software, to name a few) that will serve to unfold the mysteries of GCR. The remainder of this section will deal only with the functional commands.

After the GCR editor has been selected and loaded a

menu is displayed and the user is presented with the following options:

Scan headers
View whole track
Read data block
Write data buffer
Edit data buffer
View track buffer
View header buffer

Selection is made by using the cursor keys to position the reversed bar over the option wanted, then press the return key. F1 and F3 are used to speed up the cursor movement.

GCR MAIN MENU COMMANDS

ln2
SCAN HEADERS. This option lets you to see the headers on a track (and selecting one allows you to avoid having to manually enter the header information when using some of the other options). First the number of blocks is stored at the beginning of data followed by the bit density of every block (range is 01 to 04). The next 200 or so bytes will be 00 followed by the headers for the track. To select a header, highlight it with the cursor keys, and press return. Active keys are:

Command	Function
Cursor up....	Move up slow.
Cursor down..	Move down slow.
Return.....	Select a header.
F1.....	Move up fast.
F3.....	Move down fast.

The header that you select will be used as a reference for the program to locate the disk data when you elect to look at a track or sector.

When viewing the GCR window, the first byte is the number of header and sectors on the track (30-40 is normal), the second is a constant (normally 42 or ascii 'B'), and the following bytes indicate the block density (sequential order) in the range 01-04. The GCR bytes are listed in hex values.

Viewing the Hex window, you see a hex conversion of the GCR values in the window above. Normally the first byte is the header block (usually 08) followed by parity (or checksum) byte, sector and track. As the data is scanned, the first byte in the window may be a data block byte (usually 07).

IMPORTANT NOTE - the data in the hex window is valid until a non-GCR byte is encountered. Due to the 4/5 conversion ratio between HEX/GCR the window will display incorrect data from the point where the non-GCR byte was encountered.

VIEW WHOLE TRACK. This option will allow you to view an entire track but you will need to first scan

the disk and select a header. Optionally, you can select this mode and type in the header bytes (if you know them). The recommended minimum number is 3 as anything less may cause the drive to find several starting points for reading (which has a propensity to display incorrect data). Active keys are the same as for the SCAN HEADER except that the RETURN key exits from viewing and returns you to the menu.

READ DATA BLOCK. This option first prompts for the track number (in HEX), then the data block, that you wish to view. Next prompt is for a selected header and if you choose that option, you will be prompted for the number of bytes (8 is the average). A number less than 3 will probably give you the wrong data. The next prompt is for the number of sync bytes (to skip) which normally is 0. The last prompt is for the type of bit density to use for each header (the range is 01 to 04). After hitting return the data will be read and you are ready for the EDIT mode.

EDIT DATA BUFFER. After entering the Edit mode you will have the following commands available.

F1 -Return to main menu.
F2 -Pattern fill. Use to create repeating patterns.
F3 -Delete bytes. All bytes highlighted are deleted.
F5 -Insert bytes. Up to 255 bytes may be inserted.
F7 -Aborts any of the above (except F1).
Ω -Jumps to the end of the buffer.
CLR-Jumps to the start of the buffer.

Notes on viewed data. When viewing the data buffer, sync marks are represented by a value of $13 (ascii 'S'). If a $00 (illegal GCR coding) is embedded in the data, this byte tells the drive that the next three bytes are operating system data. Following the $00 the first byte represents the bit density to be read/written (A density of 1 is $00, 2 is $20, 3 is $40, and 4 is $60). The second byte denotes the number of times to repeat the byte that follows (third byte).

The embedded $00 is used to change bit density in the middle of a block or to "zero" bytes written to the disk. If the bytes $00 to $0F are written to the disk they will display as random bytes when viewed.

Before writing to a disk with this editor you must go through the data in the buffer and change all the sync bytes ($13), using the $00 format to enter $FF in place of the $13, $FF being the actual disk sync byte. To end the writing you must enter four $00 bytes after your altered data. This MUST be done or a drive lock-up may occur.
Example: 00 20 05 FF will write 4 sync bytes at bit density 2. (Remember: 5 GCR bytes = 4 HEX bytes which is the reason for the 05 value.)

WRITE DATA BUFFER. After editing the information in the data block, this option allows it to be written to the disk. You will be prompted for a filler byte to be used and any value entered at this point, other than 00, will be written to the entire track

before the buffer information is written.

VIEW TRACK BUFFER. This option allows you view the GCR data of an entire track. Cursor and F1/F3 keys work as in the header scan mode.

VIEW HEADER BUFFER. Lets you select another header without entering all the prompt data. Commands same as in viewing a track buffer.

ADDITIONAL NOTES.

All of the GCR editor user input data must be entered in HEX.

When entering a track at a prompt, a decimal will be printed and a 0 displayed. The 0 may be changed to a 5 to specify a half-track.

The F7 key will abort and return to the main menu at any prompt.

SUPER DOS tm

There are three versions of Super Dos fast loaders on side one of your Super Kit disk. There are some differences in these versions which allow you to select the one most suited to your needs.

Super Dos 1. The swiftest version, resides in memory from $CB00 to $D000 (all versions occupy this area). The screen is blanked and interrupts are turned off during loading. Files must be in Super Dos format for optimum efficiency.

Super Dos 2. Screen continues to display but the interrupts are disabled. For fastbooting normal Dos files. Super Dos files will load even slower than normal files with this version.

Super Dos 3. Both screen and interrupts enabled. Slower than Super Dos 2 but still much faster than normal disk access. This is the Dos the C-64 should have come equipped with from Commodore.

Super Dos can be loaded and then activated with SYS 52992 (or a JSR $CF00). Now type NEW to reset pointers before attempting to load a file, otherwise you will get an 'Out of Memory' error. Once enabled, files can be quickly loaded to all memory locations except the cassette buffer. The cassette buffer is used as a relocatable area to allow loading programs that use the area normally occupied by Super Dos. Upon loading a program that occupies the high memory area where Super Dos resides, you

will need to disable Super Dos before running the program. It is disabled with a SYS 65418.

Look for the **SUPER DOS CARTRIDGE** which will include Super Dos versions 1 and 2, SUPER SAVE™, and a multitude of other features...in the near future.

AUTOBOOT CREATOR

This program is not found in any of the menus but is contained on the front side of your Super Kit.

Load the program by typing LOAD "AUTOBOOT*",8 and then 'RUN' it. It is prompt driven and requires only that you supply the name of the program you wish to autoboot, your choice of the Super Dos version to use, the load (or JMP) address in hex if other than a basic program, and a name for the autoboot file. The load address for programs that do not load at $0800/2048 can be determined by the SYS number used to activate the program. Just convert the SYS number to hex and enter it at the prompt (the monitor in the Sector Editor can be used to obtain this conversion).

If part of your program loads into the $CB00-D000 area, normally occupied by Super Dos, then Super Dos will have to be disabled before the program will function properly. In this case you will need to insert a line into your program to disable Super Dos with a SYS 65418. When using this bootmaker, remember to filecopy the Super Dos you selected onto

the same disk containing your program and autoboot file. Your program will gain additional load speed if file-copied to the disk in Super Dos format.

THE SUPER KIT SECTOR SURGEON

Sector Surgeon may be loaded via the main menu on side two of your Super Kit or by using the (:*2) shortcut load feature. It will not be necessary to make a copy of your original prior to using this program as Sector Surgeon will make the duplicate copy in addition to performing the parameter surgery. Sector Surgeon will prompt you to insert a source to duplicate and then prompt for a blank (destination) disk. Just follow the prompts as you would with any disk copy program. You perfectionists may wish to use the Single Normal copier on your duplication to remove any Dos errors contained on the disk, but it will work fine without this extra step. (Post-Op word: The Surgeon does not wipe the disk of Dos errors, it merely removes or bypasses the protection).

Should you encounter a problem with getting a copy to run after using the Surgeon, you should FIRST make a copy of your original program. Now load Sector Surgeon and make a duplicate of your COPY. It may be that your drive is slightly out of alignment or the original may contain one or more "garbage" sectors. The Single Drive normal copier does not reproduce any errors and the copy made with it will be "customized" to your drive. Now, when

you hand it to the Surgeon, it will know exactly where to perform the delicate surgery to provide you with an operational duplicate.

Note: There are some programs that use a non-standard Dos which, by the very type of format, affords a measure of "protection". Duplication of this type of program cannot be said to be "completely" unprotected although normal Dos errors and error checking are either removed or bypassed to provide a working copy.

This option is made available in the interest of saving time as well as eliminating the head-knock abuse encountered in some DOS protection schemes. You may also discover that some of those previously heavily protected programs, on which you performed this delicate surgery, will not only be easier to boot but possibly run on that drive you thought was incompatible with the program.

SUPER SCAN

Super Scan may be selected from the side two menu or by loading directly with LOAD"*3",8,1 which is the short-cut load time-saving feature.

To perform an error check on the disk to locate non-standard data, press the 'E'. A scan of the complete disk should take about 30 seconds or less. If the first few tracks or the entire disk displays Dos 20 errors then you should turn your drive off

and back on, then re-scan the disk (some versions of the 1541 fail to initialize properly).

The bit density scan is activated by pressing 'D' and should take about 15 seconds.

After viewing a disk you can return to either scan option by pressing the F1 key.

SCAN DATA INTERPRETATION. As the diskette is scanned, the data contained on each sector is analyzed, then reported as a single symbol corresponding to that sector. Each track and sector is labeled on the screen layout.

ERROR SCAN:

Symbol	Interpretation
.	A normal sector. Contains only sync bytes.
0	Dos 20 error - can't find the block header
1	Dos 21 error - no sync (unformatted sector).
2	Dos 22 error - can't find the data block.
3	Dos 23 error - data checksum wrong.
7	Dos 27 error - header checksum wrong.
9	Dos 29 error - disk i.d. mismatch.

DENSITY SCAN:

Symbol	Interpretation
1	Normal density for tracks 1-17
2	Normal density for tracks 18-24
3	Normal density for tracks 25-30
4	Normal density for tracks 31-35

You will quickly learn that Super Scan is not your "run of the mill" error-checking program.

APPENDIX

APPENDIX A. Problem Solving.

These are some suggestions that you may try if you should have problems with any of the following:

Problem	To correct
Super Kit Disk. Fails to load.	Make sure you have removed ANY cartridge or device from the cartridge (game) port. Turn on all drives that are connected in-line with your computer. If it still fails to load and your printer is connected, try turning it on before loading.

Error Scan. Incorrect read. Turn off your drive, then turn it back on and re-scan the disk. Several tracks of Dos 20 errors at the start of a scan is an indication that the scan is incorrect and the drive must be re-set.

Sector Surgeon. Copy won't work. Make a copy of your original with the Single Normal copier. Now make a copy of the normal copy with Sector Surgeon.

APPENDIX B. Notes.

There has been alot of time and effort placed into this package and we at Prism Software would like to thank those that helped us design/de-bug or suggested more things to be added.

We feel that this package is a good bargain along with being one of the most powerful to date. Rest assured that we will continue to put out more quality products that inncorporate power/innovation along with keeping a close eye on value. This will not be limited to utilites as we do plan to deversify very soon.

If you feel that you have written a program that is marketable we suggest that you send us a non-disclosure agreement, which will fill out and

return to you, then send us the program and if it meets our standards we will be more than happy to market the program for you. We pay a very high royality and we are set up for national distribution.

We would also like to thank YOU, THE BUYER. for this purchase without your we would not be here.

Thanks to these folks for all thier help/support over the last year:

Mr. & Mrs Martin Franz	Felix Rivera
Jim Butterfield	Jim Ray
Willaim Tye	Jim Oldfield
Larry Phillips	Deepak Midah
Mindy Skelton	Libby Moorehouse
Horst Hlufhko	Steve Willaims
Brian Adamik	Brian Ingram
Steve Leuschner	Paul Richards
Tom Dash	Lynn Baxter
Fred Aquilar	Dr. Richard Immers
Richard Evers	Karl Hildon
Chris Zamara	Leroy Hooker
Jessie Knight	Matthew Leads
Harold Seely	Mike J Henry
Rob Vaughn	ALL HOTCHUG members

Thank you and we look forward to working with more of you in the future.

James Domengeaux
Pres-Prism Software Inc.

THE HISTORY OF COMMODORE 64 PROGRAM PROTECTION

W r i t t e n b y R o b V a u g h n

Chapter 1

Random Files - Disk Data - Disk Errors - Data Under Errors

When the Commodore 64 first appeared, the only company who knew any way to protect programs was Commodore itself. This protection consisted of cramming games into cartridges, which at the time, were assumed to be "unbreakable". This soon proved untrue.

Early Protection Schemes.

One of the first REAL games released for the Commodore was "Jumpman". The original version consisted of unprotected program files. At the time, the only readily available copy program was "1541 Backup", a BASIC copier that took up to 40 minutes to copy a disk. Think of the last time you spent 40 minutes copying an ENTIRE BOX, and I think you will get the picture. Next came the InfoCom adventure

games, which were also unprotected, but used unvalidated random files to store data, which confused many early users (and still confuses a few newcomers).

The first form of copy protection came with "auto-running" loaders checking the disk for certain information before running a program. Even the simplest copy program handled this with ease, but it was the introduction of "Micromon" that allowed hackers to actually unprotect the programs. "Micromon" is a machine language editor, much like "Supermon" and "HesMon".

DISK ERRORS.

The first disk error protection came from Commodore, used on "Easy Script". It was a simple error on one sector of the disk, and the error channel was read after a block-read before the program started. At the time, the error was impossible to reproduce (although we NOW know that is certainly not the case).

Months later, an obscure German programmer came out with a program that brought the 1541 into the true world of computing (and copying). This was "S-Copy", a four minute backup program that could actually detect and reproduce error 23 (WOW!). Thomas Tempelmann received almost nothing for his work, and "S-Copy" (also known by various other titles) was considered public domain and given away freely.

The next stage of protection was to go beyond error 23, and to reproduce all possible read errors that could occur. Loaders were made to search several sectors and tracks looking for different errors, and in some cases, all unused tracks were filled with some form of error. When such errors are read without proper precautions, the drive attempts to re-align the head by pulling it back 45 tracks, regardless of the current position. This creates the annoying and damaging drive "clatter". I personally lost my first drive to mis-alignment due to constant head banging (not the Heavy Metal kind either) and to this day will cringe at the sound of the drive head attempting to work it's way out into my lap. Almost all protected programs now have precautions that prevent the head from banging, but It has cost countless users their drives which, to a real computer addict, is a fate worse than death. The newest versions of the drive have reportedly been designed to prevent this problem. Some companies, for a rather large sum, can also take your old drive and rig it with special stoppers and such to prevent it from becoming mis-aligned.

Many loaders have used errors to calculate formulae to decode information on the disk, but more on that later.

DATA UNDER ERRORS.

The next phase of protection was when someone discovered that, through some DOS manipulation, sectors and tracks could be read regardless of

errors that they contained. Information could be stored and errors written over them to "hide" them from the current wave of copy programs and sector editors. This is easy to understand. Say a sector has a checksum error on it (Error 23). The normal DOS routine assumes that the data on the block is incorrect, and refuses to read it (and also slaps your read head around a bit). But if a special routine is written in disk memory (also known as buffer RAM) the job queue can be used to read the block without normal checksum checking. Thus important code and data can be stored "under" an error. Types and descriptions of read errors are as follows:

ERROR 20: Block Header Not Found.
Caused by lack of header block or an attempt to access an illegal sector.

ERROR 21: No Sync Character.
Caused by lack of sync mark, head mis-alignment, or by an unformatted disk.

These errors, when data under error protection first appeared, were too complex to use. Modern DOS manipulation can easily access these types of data blocks through timed head reading.

ERROR 22: Data Block Not Present.
Caused by incorrectly written data, lack of data on disk, or illegal track/sector requests.

I have yet to encounter an Error 22 that actually

contained usable data. It was one of the last error types to be produced artificially, and was used exclusively for error protection.

ERROR 23: Checksum Error In Data Block.
Caused by grounding problems, or write errors. The checksum for the block does not match the calculated checksum from the data present.

ERROR 27: Checksum Error In Header.
Caused by grounding and formatting problems. The checksum for the header does not match the track/sector calculated checksum.

These are the most easily produced, as all the data stays intact. All that is necessary to change is the checksum itself, or just one byte in the data block or header through DOS manipulation. Data is easily read using job queue routines. It is also easy to use a track repair program to reveal the data "beneath".

ERROR 29: Disk ID Mismatch.
Caused by a bad disk header or a non-initialized disk. The ID in the header does not match the calculated ID.

This is frequently used, and although not as easily done as Error 23 or 27, data remains undisturbed.

Most copy programs support these types of protection, and several utilities have excellent sector editors that read data under errors, and even

repair them while keeping the information intact.

SINGLE FORMATTED TRACKS.

An unformatted track will produce an Error 21 when a block-read is attempted. Some programs are stored on the disk with only the used tracks formatted. A normal copy program (most of which initialize a blank disk before the copy process begins) will read an unformatted track as an Error 21, and create the error on the formatted block. A good program, through timed DOS manipulation, can detect the good block or the lack of signs that the track is in an unformatted state, and thus tell if the version, although the errors are correct, has been copied..

CHAPTER 2

Extra Tracks, Half-tracks, Embedded Tracks and DOS Manipulation

In chapter one I discussed such concepts as random files, disk data, disk errors, and data "under" errors. These protections, once at the height of use, are now considered quite old even though they are still common. They can be reproduced by almost all the modern copy programs, and many good

utilities have functions to eliminate and reproduce such protection schemes. Now, let us look at some of the most current protections in wide usage.

EXTRA TRACKS.

The concept of extra tracks is very easy to understand. A standard format disk has thirty-five tracks, from the outer (track 1) to the innermost (35). However the innermost track is not quite next to the center hub, thus leaving room for extra tracks beyond. The head, through DOS manipulation, can be positioned outside of the last track, and used to read and write data and/or sync marks. The stepper motor is capable of moving "in" to track 40, but the only tracks safe for data storage are 36 through 38. Tracks 39 and 40 can contain disk errors and sync marks, which is one of the newest protections coming into use right now. Most copy programs handle copying out to track 38 (at least) and most check out to 40. Also, half tracks between these extra tracks are quite possible. Still, even data hidden under an error on track 36.5 (track 36 and then a half track) is easily copied. To move the head out to these extra tracks you need a small DOS routine that does not use the SEAK job. It moves the head out after it has reached track 35. One way of doing this is to cycle bits 0 and 1 of location $1C00 (DSKCNT) in drive memory. Rotate the bits as such, $00 - $01 - $10 - $11 to move the head inwards, and $11 - $10 - $01 - $00 to move it outwards, all in half track increments. One small word of caution on this subject. By attempting to

move the head beyond 40 (and even 38 for that matter) may cause the head to "lock" out in that position. Sometimes the simple (though painful) remedy is to "bump" the head using the BMP routine in DOS. On occasion, the head may become so "stuck" that it is necessary to take apart the drive and manually re-position the head. Your average computer user would have no idea how to do this, and would end by either buying a new drive, or shelling out bucks to get it fixed. I have actually come across programs that claim to damage equipment if the program is copied, and this was how it was done.

HALF-TRACKS.

Half-tracks are also quite easy to understand, and to use for that matter. The DOS itself provides a routine to move the head in this fashion. For the most part, data can even be read right down the disk as you would data from normal tracks. Half-tracks may contain the whole shmear of protections, errors, data, data under errors, and sync marks. But half-track data is not very safe, as the head tends to "overrun" as it writes, and may spill garbage bits onto the normal data tracks next to the half being written. Half-track data is best put on a disk between two tracks that contain no data, and is usually written last. A sports program from a well-known software company, extensively used half-tracks, not for protection, but for data storage. All the data had to be stored in sequential random files to be read by the superfast loading

routine they used. Two events were also stored on the disk so that the regular slow load routine would not have to be used. All this data would not fit on a single disk, so they resorted to extra tracks AND half-tracks. In early versions, the first event would not function due to data mixing. As you can see, half-tracks are not always reliable, and double-density disks should be used.

The drive head can be moved by altering two bits in location $1C00 (DSKCNT) which controls the stepper motor. The regular REED routine will read data from the disk, no matter where the head is positioned. (Please note: REED, SEAK, and BMP are the official names for the DOS Read, Seek, and Bump routines.

Many copy programs can handle both extra and half-track protections (or data storage, in some obscure cases). There is at least one program which contains a half-track reader and formatter. The reader only reports the existence of half-tracks (reads the error channel). I have often come upon half-tracks that read as errors, but certainly not the usual non-format error. This is a clue to definitely copy half tracks.

EMBEDDED TRACKS.

Embedded Tracks, although peculiar, play an almost non-existant role in copy protection today. It certainly is at the rout of the new "Fat Track" protection, as the two are very similar.

An embedded track simply modifies a track so that the next track (i.e. modify five, and it effects six) will write to a totally different track. By making use of an embedded track formatter, the results to a normal sector editor are as follows:

Track modified: 5 Embedded track: 17

Data on track six is replaced and an Error 21 occurs when attempting to read six with a normal sector editor. Track five is untouched, and is normal. Track seventeen is also normal. If you attempt to write to track six using an error-ignorant sector editor, any data written is instead written to track seventeen.

The purpose of embedded tracks is pretty much this...by DOS manipulation, data can be read to the "hidden" track and the data cannot be modified in any way, such as trying to change protection routines. Any data written will produce an error in normal sector editors, and if an error-ignorant one is successful in writing, the data will be written to the embedded track. If one used track eighteen for the embedded track, any successful writes would ELIMINATE the BAM.

This is the only use I see for embedded tracks, and I have yet to encounter one used as protection. They are very interesting though, you must admit.

DOS MANIPULATION.

This seems an appropriate place to reveal some

information discovered by a good friend of mine. The putz finally obtained an INSIDE COMMODORE DOS book and, by using the scads of free time available to him, has came up with some beauties. First let me clarify the disk commands used.

Memory Manipulations:
PRINT#15,"M-W"CHR$ (low byte) CHR$ (high byte) CHR$ (number)...CHR$ (data)
PRINT#15,"M-R"CHR$ (low byte) CHR$ (high byte)

Block Read:
PRINT#buffer,"U1:"buffer;drive;track;sector

Block Write:
PRINT#buffer,"U2:"buffer;drive;track;sector

By decreasing the Timer 1 High Latch in the 6522 Disk Controller (i.e. lower value in location $1C07) one may speed up the head movement, around six times as fast, by this command:
OPEN15,8,15:PRINT#15,"M-W" CHR$(7) CHR$(28) CHR$(1) CHR$(15)
Any value less than 15 in the CHR$ (data) byte will cause the stepper motor to jump gears and move, while the head stays immobile. This is how "drive head music" is done. This is VERY dangerous, and should NEVER be done!

By speeding the head up with the above command, certain disk functions are sped up accordingly. A LOAD or SAVE, which moves the head little, is unaffected. The VALIDATE command, and NEW command

are cut in one third. A validate that took 135 seconds can be done in 45 with this modification.

The fast-bump that occurs when NEWing a disk after this modification is harmful, but no more than the normal bump is. Many programs that affect the drive speed do not utilize this, and may be sped up even more. "9 Second Format" is an example, but speed changes are negligible.

****
* CHAPTER 3 *
****

Sync Manipulation, Extended Data and Header Blocks

In Chapter Two, I went over the uses and descriptions of such schemes as extra and half-tracks, embedded tracks, and drive head speed. Embedded tracks will be discussed more thoroughly when I cover Fat Tracks. The said protections are nothing to your average Gee-whiz copy programs.

SYNC MARKS.

The first thing that must be explained is what a sync is, and the normal usage. Commodore, not trusting their peripherals to act like real ones, has (as you may well know) always taken the "safe" approach to everything regardless of time and space

factors. Instead of allowing the computer and drive to handle timed data inputs (i.e. read data when a timer goes off) they use special bit arrangements to signal the beginning of header and data blocks, which together make up each sector. A sync mark is [*quote:I.C.D.] "a very special character consisting of 10 or more 1-bits in a row, normally 40 of them". In other words a sync (minimum of 10) looks like this: 1111111111, on the disk. Why forty?.. Well, the GCR conversion is far easier when done in groups of FIVE bytes. GCR code will be covered in a later chapter so you will just have to take this for granted.

When the drive is reading the disk (yes, it reads byte by byte) and comes across ten or more 1-bits, it knows that coming up is a header or data block. The first zero bit it encounters starts it storing data, and depending on the first byte, it reads and stores a header or data block. If a $08 is encountered as the first byte, it signifies that the block is a header. $07 signifies a data block.

When writing syncs, the DOS is in sync mode, which means that the five $FF bytes are not translated into GCR code.

Now that you understand the basic use of a sync, let us look at how the header and data blocks are arranged.

HEADER BLOCK SETUP.

SYNC MARK: A group of ten or more (up to 2,611,200) 1-bits in a row that signal an upcoming header or data block.
HEADER BLOCK ID: A $08 byte, signifying a header block.
HEADER BLOCK CHECKSUM: Checksum to assure that the block was written correctly. The track and sector numbers are EOR'ed with the two ID characters to arrive at this byte.
SECTOR NUMBER: The sector for the next data block.
TRACK NUMBER: The track for the next data block.
ID CHARACTERS #2 AND #1: The two bytes specified in the formatting of the disk. Note that ID character #2 is the second of the two byte ID you typed in before formatting.
TWO $0F BYTES: Used for formatting, never used again. (Though it possibly could be changed for protection purposes...possibilities arise!)
HEADER GAP: Eight $55 bytes, unread by the DOS. This allows space for the header and data blocks.

Note that the header block is written only during the format process and never written to again.

DATA BLOCK SETUP.

SYNC MARK: As explained above.
DATA BLOCK ID: A $07 byte signifying a data block.
256 DATA BYTES: Actually 320 bytes that contain

the data to be read.
DATA BLOCK CHECKSUM: Arrived at by EOR'ing the data bytes together. (EOR is a machine language term. If you do not understand the operation, refer to a book on ML.)
TWO $00 BYTES: Again, two OFF bytes used in the format process.
TAIL GAP: Four to twelve bytes at the end of the data block separating the end of the data and the next header block. This varies from track to track, due to the smaller track circumference. Again the DOS does not read these bytes, although they can definitely be changed for protection schemes.

Note that the entire data block, including sync marks, are re-written each time the block is written to. Each combination of header and corresponding data block make up one sector on the disk.

SYNC MARK MANIPULATION.

Now we arrive at the interesting part, changing sync marks for protection. Well, I certainly hope you have not just skimmed through this chapter. A working knowledge of how blocks are set up will aid you greatly as I try to explain this and extended blocks.

The easiest sync to change is the one preceding the data block. Why? Because, as I noted, the data block is re-written each time, including the sync mark. Now, a small change in the sync will cause no damage

to any other data block, as the above mentioned tail gap will just become smaller. If the number of syncs exceed the tail gap space, the block will over-run right onto the next header block. Because of this, many people protect the disk with syncs and no usable data on one or more tracks.

The good sync write and read utilities rest in the hands of the software companies, and deep in the code of the better copy programs. Of course, if a program can read the syncs in and check them, then it can be broken. In the Top Secret Stuff writer, the 10,240 suggested number of bytes is the exact number written to erase the disk during the format. Just type 255 for the written bytes, and you will have filled the entire track with syncs. Try entering 99,999 or so for the number of sync bytes, and then try reading the track. The drive will spin off into Never-never Land. Some programs use this as a basic scheme. Almost all of the copy programs will die on a track set up like this, unless you have copy parameters with a very good nibbler.

Changing syncs in the data block is easily accomplished by rewriting the DOS routine into buffer RAM and just altering the number of syncs written. This also holds true for reading the syncs back.

Now, the harder protection is changing the syncs in the header block, since it is only written to during the disk format. This can be done two ways; by writing a special routine that reads the tail gap

and then adds syncs to the ones present when it reaches that point, or by using a self-styled formatter. A fast format program has to write these syncs when doing it's job, and by changing the number for every sector, or just one or two tracks, one can actually protect the disk before the program is even written to it. A good one would actually be able to go through and reformat the header blocks without touching the data, and I think it would not be too hard to pull off.

This should give you more than a basic understanding of what syncs are and how they work. When copying, if all your programs seem to choke at some point, try reading the disk with an error checker. The track on which it falls apart, there is probably (well, more than probably) a full track of syncs. Use a copy program that allows you to skip tracks (there are a few) and then use a sync writer to add 10,240 syncs of 255 ($FF), which is 2,611,200 1-BITS. Usually this will do the trick, although there are no guarantees.

EXTENDED HEADER BLOCKS.

First, is the header block. If you remember from previous descriptions, the header looks like this:

```
 Sync:Header   ID:Header  block:Sector:Track   :ID
char.:ID char.:2 $0F:
  mark:block     :checksum     :number:number:number
1:number 2:bytes:
```

Through DOS manipulation, one could extend this block to include, say, three $0F bytes after the main section. Or eliminate the track and/or sector numbers and put more than one checksum. This can be manipulated in many ways, but it is not very common. and probably will not be now that modified GCR has been discovered. These schemes are included to attempt to cover all possible protection angles. The one other major scheme will be covered while I explain extended data blocks.

EXTENDED DATA BLOCKS.

Extended data blocks are used for two purposes. Stopping copies, and/or adding more data to a disk. As we know (unless you did not read my earlier chapters very well) each track is split up into a certain number of sectors each with it's own sync mark, header block, header gap and a data block. At the end of all the sectors, separating the first from the last, is the tail gap. Through the complexities of the DOS each track is formatted to have one sync mark, one header, one gap and the rest taken up by one extended data block that has all the information about it stored in the single header. Data is read and sequentially stored in buffers until the usual block end is reached. The the drive goes on to the next track to repeat the process until the loading is finished. To be able to read the data coming in, two buffers must be used. While the drive is filling the second (having finished the first), the computer must be emptying the first

buffer. When the drive fills the second one, it wraps around begins filling the first again. Computer/drive timing and checksums must be perfect. If the computer reads too fast, it will overtake the drive and read random or repeated garbage in the buffer. If the drive is too quick, it will soon write over data before it can be read into the system. The easiest way for the drive and computer to interact is for the drive to pause very briefly (we talking micro-seconds!) and tell the computer to read the buffer it has finished with. The computer can go ahead and cram all the data into RAM before the drive finishes the next buffer, and wait for the next signal to continue.

Offhand, I can think of no specific program that does this, although the Epyx Fastload cartridge works on principles similar to this. Note also that the computer/drive process can be used to read any type of disk setup, although the lack of individual sectors will speed the drive up greatly since it can, technically, read a tracks in one rotation instead of the three to four it takes during a normal load. Regardless of the attitude one may take about the 1541 drives, they have proven to be very reliable in saving data (and the less said about alignment problems, the better).

ELIMINATION OF SYNC MARKS.

Quite simple actually. The syncs are replaced by some non-sync data, making the particular sector unreadable to your shmoe-average sector reader. The

drive is either instructed to use the replacement data as syncs, or timed to ignore them and to begin reading at an approximate correct time to catch the data byte signifying a header block. It is not anything common, although sync manipulation is a common practice. Extended data blocks would fall in this same category. It is mostly used in conjunction with modified GCR code.

Chapter 4

Group Code Recording (GCR)

What follows is a detailed, disgusting outlook on GCR format and it's place in program protection. So, without further adieu..(I've always wanted to say that!)

GROUP CODE RECORDING.

Group Code Recording, or simply known as GCR, is the process by which your 1541 drive records data to the disk. Simply stated, the purpose of GCR is to rewrite the data in such a way that no possible combination of ten or more 1-bits may appear in the data block.

As you should know (since you have carefully read the previous chapters) that a sync, used to find the beginning of each header block on a track, is made up of forty 1-bits in a row which is detected by the read head. Forty 1-bits is the same as 5 bytes with all the bits set to 1, or 5 bytes of $FF (255 decimal). Now, say you had a block of machine code, and at the end there were five or more $FF bytes. Having saved that data directly to disk, you would be (in effect) making a sync mark in the middle of your data. This will send the drive scurrying off to wherever it goes when it finds a problem it cannot cope with, commonly referred to as Never-never Land. So when the Commodore engineers sat down to design the 1541, they were aware of this potential problem and came up with a solution, the code translation known as GCR.

To prevent the problem from happening, all the bytes written to disk are broken up and translated in such a way that no data written to the data block will ever contain ten or more consecutive 1-bits. This is done by "binary to GCR" conversion. Each data byte is broken up into two nibbles, or two segments of 4 bits each. Each of these, in turn, is translated to respective 5-bit counterparts which restrict the possibility of ten or more 1 bits appearing in the data. At the same time, it provides that no more than two consecutive 0 bits will appear in the data. This provides for timing and accuracy during disk reads and writes.

Now that I have you thoroughly confused, saying "Why

the hell do I put up with this from a book?", I will run through a demonstration conversion so you can see exactly what happens when the drive writes data to each data block. First, we need a chart for converting the binary data to GCR format.

Hex data	Binary	GCR	Hex data	Binary	GCR
$0 (0)	0000	01010	$8 (8)	1000	01001
$1 (1)	0001	01011	$9 (9)	1001	11001
$2 (2)	0010	10010	$A (10)	1010	11010
$3 (3)	0011	10011	$B (11)	1011	11011
$4 (4)	0100	01110	$C (12)	1100	01101
$5 (5)	0101	01111	$D (13)	1101	11101
$6 (6)	0110	10110	$E (14)	1110	11110
$7 (7)	0111	10111	$F (15)	1111	10101

A sample conversion: given $FE (decimal 254)

Step 1. Hexidecimal to binary:
$FE (254) = 11111110
Step 2. High and low nibble split:
High nibble = 1111xxxx = 1111
Low nibble = xxxx1110 = 1110
Step 3. High nibble conversion
High 1111 = GCR 10101
Step 4. Low nibble conversion
Low 1110 = GCR 11110
Step 5. Final GCR concatenation
10101+11110 = 1010111110

Now you might wonder how this new ten-bit code can be written to a disk that stores in groups of eight

bits? Well, conversion is done FOUR bytes at a time. When 32 bits are converted, they are replaced by 40 resultant bits which is easily divided into 5 new bytes. Therefore, each data block TECHNICALLY holds 320 bytes, not 256. If this conversion was not necessary, a disk would have 64 more bytes per block available, or 41996 bytes per disk which would translate into approximately 164 new blocks.

When the disk is writing, it is taking groups of four bytes from the data and converting them into groups of five which are stored on the disk. Simple enough is it not?

Another example: given $34, $EA, $F1, $A2 - a group of four bytes.

Step 1. Hex to binary conversion

$34 = 00110000
$EA = 11101010
$F1 = 11110001
$A2 = 10100010

Step 2. Nibble high/low conversion

00110000 = 0011 + 0000
11101010 = 1110 + 1010
11110001 = 1111 + 0001
10100010 = 1010 + 0010

Step 3. Binary to GCR conversion

0011 = 10011 0000 = 01010
1110 = 11110 1010 = 11010
1111 = 10101 0001 = 01011

```
          1010 = 11010   0010 = 10010

Step 4. GCR codes
          10011 + 01010 = 1001101010
          11110 + 11010 = 1111011010
          10101 + 01011 = 1010101011
          11010 + 10010 = 1101010010

Step 5. Concat and split          New GCR bytes
10011010 and carry 10          =  10011010  byte 1
10+111101 and carry 1010       =  10111101  byte 2
1010+1010 and carry 101011     =  10101010  byte 3
101011+11 and carry 01010010=     10101111  byte 4
01010010                       =  01010010  byte 5

Step 6. Binary to hex conversion
          10011010 = $9A (decimal 154)
          10111101 = $BD (    "   189)
          10101010 = $AA (    "   170)
          10101111 = $AF (    "   175)
          01010010 = $52 (    "    82)
```

The result is that four bytes $34, $EA, $F1 and $A2 become five bytes $9A, $BD, $AA, $AF and $52 which prevent more than ten 1-bits in a row and allow no more than two consecutive 0 bits in each byte. This precludes the possibility that you will accidentally write a sync mark in your data.

READING AND WRITING GCR CODE.

I hope that you understand that when your drive is READING, it is converting all the data from GCR to

binary and does the opposite, i.e. converts all the binary to GCR data, when WRITING. The only time the drive does not do this conversion is when it is in SYNC MODE but that is the only exception.

When the drive reads a block, it puts the first 256 bytes of GCR data in a buffer, and puts bytes 257 to 320 in an overflow buffer. Then the data is converted back to binary and put in the buffer being used for that particular disk read. When writing, the reverse of this process occurs.

GCR PROTECTION SCHEMES.

Simply put, the conversion table is different from the one outlined above. The data can be read from disk, but comes out looking like garbage. A complete revision of the table can be done, or just two or more interchanged. This would result in more trouble than it seems, since each of the four bytes is broken up into two nibbles. If only two are different, much of the code would change.

To pull "modified GCR code" off, one simply re-writes the GCR to binary conversion routine in DOS to the RAM buffer. Then the read routine is re-written, using jumps to the main routines (they can still be used) to read the data in. But then the new conversion chart is substituted, and the rest is entered as it would normally be. This is an easy procedure that provides very complex protection. Some of the newest copy programs are not stymied by this, reading the GCR and copy it over itself.

Remember this, if the program can read it, so can you.

A technical "possibility" scheme. Although I have never seen this it should be possible to write a track set up with one sync mark, a couple of different checksums in a larger header and one long non-GCR converted data block. With the proper "read and fill" slam, bang fastload routine, a single program could be stored on a couple of tracks. All the data would be unreadable except by the program. I feel sure that someone will tackle this somewhere down the line. Normal sector editors and loads would be useless, only reading bad data or encountering errors. Whether protection schemes continue in this direction remain to be seen.

WARRANTY AND LIABILITY

LIMITED WARRANTY. It is the responsibility of the purchaser to determine the suitability of this product for his or her purpose. Prism Software, any dealer or distributor makes no warranty, express or implied, concerning this product or any portion thereof, including the accompanying manual. Prism Software will not be liable for any damages resulting from any defect or omission in this product, or any portion thereof, nor be liable for any damages relating to, but not limited to, any loss of business, interruption of service or other consequential damages occurring from the sale of this product. Prism Software reserves the right to improve and/or correct their product, or any portion thereof, at any time, without notice, and with no responsibility to provide such changes to any and all prior purchasers.

401 LAKE AIR DR., SUITE D • WACO, TEXAS 76710
(817) 751-0200

Brought to you by:

https://www.facebook.com/groups/commodoreinternationalhistoricalsociety

commodore international historical society